FluidStream

Version 7.00

Sytem Theory and Design

by Dr Roger Nokes
February 2007

FluidStream

Version 7.00

System Theory and Design

February 2007

Department of Civil Engineering
University of Canterbury
Christchurch, NZ
roger.nokes@canterbury.ac.nz

Design and layout: Melody Callahan

1 Introduction to PIV/PTV

1.1 General principles

The opportunities to gather detailed field information in fluid flows have mushroomed since the advent of digital video/camera technology. Typically in the past, quantitative velocity measurements have been obtained using “point” probes, that measure the velocity at a single location in the flow. These probes could be intrusive, as in the case of pitot tubes, hot-wire and hot-film probes and acoustic Doppler velocimeters (ADVs), or non-intrusive, as in the case of laser Doppler anemometers (LDAs).

Qualitative techniques for visualising flow structures have existed for many years. An example of which is the release of smoke, or some other tracer that follows the trajectory of the fluid particles, that can be filmed or photographed. Long exposure photographs provide informative pathlines when used in conjunction with well-lit particles in the flow.

In the last 10 or 20 years two new experimental techniques have been developed for the quantitative measurement of two, and sometimes three, dimensional velocity fields using analogue, or digital, video technology together with the processing power available through modern personal computers. These techniques are generally known as particle image velocimetry (PIV) and particle tracking velocimetry (PTV) (see Smits and Lim (2000), Raffel, Willet and Kompenhans (1998) and Blackett (1994)). The scope of these techniques has broadened significantly in recent years due to advances in computer, and digital video, technology and their practical application has increased while their associated costs have decreased.

Both techniques are based on the principle of capturing video images of a specially illuminated particle seeded fluid flow, and from that video record extracting quantitative information about the flow field. The processing of the video frames is automated, using specially designed algorithms, and executed on a computer. By far the most popular of these techniques has been PIV, although there are good arguments why PTV is equally worthy of attention. The book by Raffel et al (1998) provides an excellent introduction to all aspects

of PIV. PTV is only mentioned in passing, although Cowen and Monismith (1997) suggest that PTV offers significant advantages in accuracy over traditional PIV methods. They describe a hybrid system based on both PIV and PTV techniques.

In simple terms PIV is based on the cross-correlation of the intensity fields in two consecutive frames, with fluid velocities obtained from peaks in the cross-correlation field. For this process to work effectively the particle seeding must be sufficiently dense for the cross-correlation to be meaningful. Frames taken from the video record can be processed directly to obtain the desired flow information.

On the other hand PTV is somewhat more complex. The video frames must be processed in order to identify the particles within each frame. Particles are then tracked from frame to frame in order to produce particle-centred velocity estimates. These velocities may be interpolated onto some regular grid to obtain Eulerian velocity information, although one of the advantages of PTV, the ability to generate Lagrangian flow statistics does not require this final step. Particle density is less of an issue with PTV than PIV. In fact, sparsely seeded flows are often easier to analyse as the tracking of particles from frame to frame is relatively straightforward. However more densely seeded flows can also be processed using PTV. In this aspect, at least, PTV is more flexible than PIV.

A more detailed view of PTV will be provided in the next section.

1.2 Detailed review of PTV

The process of obtaining velocity fields for a fluid flow using PTV techniques can be broken down into four steps. These are

- Image capture
- Image processing – particle identification
- Particle tracking
- Velocity field generation.

Each of these steps is discussed in more detail in sections 1.2.1 – 1.2.4, both in a generic way, and with a particular focus on the *FluidStream* system.

1.2.1 Image capture

Image capture involves a number of issues. These are

- Lighting
- Particle selection
- Frame capture

1.2.1.1 Lighting

For two-dimensional PTV work a sheet of high intensity light is needed to illuminate the particles in the flow of interest. A range of light sources can be used, and the selection will depend somewhat on the particular requirements of the experiment.

The cheapest, but least flexible, light sheets are generated from white light sources such as a slide projector or high-intensity halogen bulb. A slit arrangement is generally required to provide the sheet, and due to the nature of white light the generated light sheet will diverge somewhat over the flow domain. The degree of spread, and the implications for the experimental results, will depend on the geometric configuration of the lighting system and the nature of the flow itself. The sheet thickness is generally easily changed for such a system by altering the slit size. However, intensity attenuation, due to spreading, can be an issue over long distances. One of the biggest challenges with a white light source is achieving high enough light intensities to allow the video capture equipment to “freeze” the motion in the flow.

Laser light sheets are commonly used in PIV and PTV work. High power lasers, typically 1-2W, are used, but at the time of writing, such laser systems are still very expensive. Laser generated light sheets have a number of advantages over those obtained from white light sources. In particular, the laser sheet is generally very thin, the order of 1 or 2 mm, and because of the ability to collimate a laser beam, the spread angle of a laser sheet is very small, normally about 0.5-1 mrad. Pulsed lasers with the ability to produce very high intensity light for a very short time, can be used to “freeze” high speed flows.

The FluidStream system can be used with any type of light sheet. However, particle motions perpendicular to the sheet may cause particles to remain in the sheet only for very short times, and this is

obviously undesirable from the software's perspective, as it is attempting to track particles from frame to frame. Therefore out of plane motions may be a consideration in light sheet selection.

1.2.1.2 Particle selection

In water based flows pliolite VT resin, a white material used in the rubber manufacturing process, has become the standard material from which to produce the seed particles. The resin has a density roughly 3% greater than water. It can be ground into particles of diameters ranging from a few microns up to hundreds of microns. Very small particles have fall velocities that can be ignored in most laboratory flows. In some flows it is possible to add salt to the working fluid in order to ensure that the particles are neutrally buoyant, somewhat removing the restriction on the size of particles used.

Particle size selection is determined by a number of factors. First and foremost the particles must act as tracers that accurately replicate the velocity of the fluid particles surrounding them. Raffel et al (1998) provide a crude estimate of the response time of such particles to fluid accelerations. This is given in equation (1).

$$\tau_s = \frac{{d_p}^2 \rho_p}{18\mu} \qquad (1)$$

where d_p is the diameter of the particle, ρ_p is their density, μ is the viscosity of the fluid and τ_s is the relaxation time of the particle velocity in response to fluid accelerations. Clearly the shortest timescale that can be accurately measured using PTV will depend on the particle size.

Particle size also affects the intensity of the reflected light, as seen by the video capture system. The intensity of the light sheet increases in importance as the particle size decreases, and in most experiments a compromise between particle size, light sheet intensity and exposure time on the camera system is required.

Introducing such fine particles into a fluid flow can be difficult due to surface tension effects. Normally the particles are mixed into a small amount of the working fluid together with a surfactant to

reduce the surface tension, before being introduced to the experimental system.

The density of particles in a flow again depends on the flow being investigated and the type of information required. Issues of seeding density will arise again in later sections.

Another possible application of PTV is in tracking surface drogues. Materials other than pliolite resin are commonly used as drogues. In particular drogues are generally larger objects, that may have a submerged “vane”. In the Fluids Laboratory at the University of Canterbury golf tees have been successfully used as drogues as have nylon particles of a few millimetres in diameter. However, surface tension can affect the motion of the drogues when they come close to a solid boundary, or one another, and this must be considered when analysing drogue data.

The *FluidStream* system has been tested in water flows using Pliolite VT resin for internal flow fields and drogues for surface fields.

1.2.1.3 Frame capture

A large range of frame capture equipment is currently available. Digital still and digital video cameras are most commonly used.

Modern digital still cameras generally provide a low resolution, digital video mode, as well as a slow automated sequential frame capture capability.

Digital video cameras produced for the mass market are perfectly good tools for image capture in the laboratory provided the lighting and resolution demands are not too great. The modern digital video standard (PAL) has 720 pixels per video line with 576 lines per frame. Unfortunately, most cameras on the consumer market today only capture video in interlaced mode (ie two half frames, 1/50th of a second apart, are combined to give a complete frame every 1/25th of a second). Progressive scan, or non-interlaced recording, whereby the whole frame is grabbed at once, used to be common in the more expensive cameras, but nearly all manufacturers seem to have discontinued this feature, apparently through lack of consumer demand. For PTV work progressive scan is highly desirable, particularly if the flow speeds are high.

One of the most attractive reasons for using mass market video cameras for video capture is not only their value for money, but the fact that software for capturing images directly onto computer, and manipulating these images, is readily available. Adobe's Premiere application is used in the Fluids Laboratory at the University of Canterbury for some video data acquisition. A digital video camera is connected through a FireWire (IEEE 1394) cable to the computer, and Premiere is used to control the camera remotely.

Higher resolution digital video cameras are available for more specialised scientific work. These cameras are colour or grey-scale (8, 10 or 12 bit) and have resolutions up to and beyond 2000x2000 pixels. The software for controlling these cameras is specialized and dedicated framegrabber cards are generally required.

One of the chief concerns in choosing a camera for PTV work is the camera's ability to work with low light levels. In general, we have found that digital video cameras provide much better performance than digital still cameras. However, even within the digital video camera range, the low light performance varies significantly. Manual control over focus, frame speed and f stop are crucial for good PTV work.

A range of digital image formats is currently supported by cameras and associated software. For example digital video formats include AVI, Quicktime, and MPEG. Digital still images are normally stored in TIF, JPEG, GIF, PNG or BMP formats.The FluidStream system provides support for sequences of JPEG, TIF, PNG, BMP and GIF files. It does not directly support any digital video format. Premiere allows the user to export a digital video as a sequence of TIF or GIF files, each frame stored in a separate file.

1.2.2 Image processing – image identification

Once a sequence of video frames has been captured these images must be processed in order to identify the particles within each frame. For each particle we wish to obtain it's location to sub-pixel accuracy as well as its intensity and size. This analysis can be quite challenging due to issues of variable intensity across and along the light sheet, varying particle size (and hence brightness) and a range of other experimental issues such as camera performance.

FluidStream provides three algorithms for particle identification that will be discussed in section 3.3.3.

1.2.3 Particle tracking

The core of the PTV analysis process is the tracking of particles from frame to frame, and it is this aspect that is the most challenging. There is a wide range of techniques that can be used, but each essentially reduces to a decision-making algorithm that must select the "best match" between particles in two sequential frames. An example of a best match algorithm is simply to select the particle in frame 2 that is closest to the particle in frame 1.

It should be noted that a match is not always possible for a number of reasons. These include

- the particle in frame 1 left the light sheet before frame 2 was captured
- the particle in frame 1 is obscured by another particle in frame 2
- the particle identification process failed to identify the particle in frame 2.

FluidStream provides a relatively sophisticated optimisation technique for matching particles between frames. A range of criteria, or algorithms, can be used to guide the optimisation process, and these algorithms can be fine-tuned by the user for their particular needs. Both sparsely and densely seeded flows can be analysed.

In should be noted that the particle matching process is imperfect. Particles can be incorrectly matched, or not matched at all. It is important that the impact of such imperfections on the predicted velocity fields is understood.

1.2.4 Velocity field calculation

The ultimate goal of a PTV system is to generate velocity fields for the two-dimensional (or three dimensional if 3D PTV is being used) flow domain being observed. The velocity can be estimated at each particle location (provided that that particle has a match in the previous or subsequent frame) by analysing the displacement of the particle between frames that are a known time step apart.

Unlike PIV, where velocities can be calculated at any point in the flow, velocities in PTV are particle-centred. This means that the velocities are known at a set of random locations in each frame. In order to deduce useful information from these velocities, for example the development of the velocity at a particular position in the flow with time, velocities must be interpolated onto a regular grid. Cowen and Monismith (1997) suggest an additional technique of "binning" the velocity measurements from all frames in order to improve this interpolation process.

The accuracy of the velocities generated by a PTV system will be limited by the following factors:

- inaccurate mapping from pixel space to physical space,
- optical distortion of the video images,
- incorrect particle identification,
- incorrect particle location determination,
- incorrect particle matches,
- missed particle matches,
- insufficient particle density to resolve the velocity field,
- inaccurate velocity estimate from matching due to streamline curvature,
- inaccurate interpolation scheme,
- inaccurate extrapolation scheme at the edge of a frame,
- particles not faithfully following the flow, and
- camera resolution.

Many of these sources of error can be minimised through careful experimental design and analysis.

Once the velocity fields have been generated, derived information, such as turbulence statistics, vorticity etc, can be calculated. However, it is important to recognise what information can and cannot be deduced accurately from the interpolated velocity fields. Turbulent properties of the flow in particular must be sought with care. The duration of sampling and the sampling frequency must be sufficient for meaningful turbulent averages to be obtained. The seeding density and flow resolution will determine the range of scales that the PTV system can resolve. In addition, the sources of error discussed above can lead to "pseudo-turbulence" that may be hard to identify. If detailed turbulence data is required it is

recommended that an LDA or ADV is used in conjunction with the PTV in order to check the tracking system's outputs.

1.3 Summary

This chapter has aimed to provide the reader with some essential background in the operation of a typical PTV system. This core information should provide a framework within which FluidStream can be understood.

2 Overview of the FluidStream system

2.1 Introduction

The general structure of a particle tracking velocimetry system has been discussed in the previous chapter. The remainder of this manual, and the accompanying, **FluidStream 7.00 – User's Guide**, will focus on the *FluidStream* system, a PTV implementation developed in the Department of Civil Engineering at the University of Canterbury, which is part of the *Streams* software library.

In this chapter an overview of *FluidStream* will be given. This will provide an introduction to key concepts in the software design as well as information about the system's operation.

2.2 User interface

FluidStream is a graphical user interface based software application written in the java programming language. All of the usual interaction mechanisms are available, namely pull-down menus, popup menus, list boxes, radio and check boxes, buttons, combo boxes, dialog boxes, windows etc.

In addition, *FluidStream* has been developed with an object-oriented user interface (see Nokes (2002)). This type of interface should provide the user with a very natural means for interacting with the data stored within the system.

FluidStream deals with a number of entities that, in the software arena, would be referred to as **objects**. These software objects represent things associated with the PTV analysis process. For example, the most important of these objects is the **video clip** (see chapter 3). It corresponds to a sequence of video frames, each containing a list of particles, from which the user wishes to generate velocity fields.

Objects are usually displayed for the user in a list box, where the object's name and its associated icon are shown.

Each object in the *FluidStream* system is manipulated in a consistent way. A popup menu can be displayed by right-clicking the mouse on the object when it is displayed in a list box. This popup menu allows the user to view information about the object, and to perform operations on it, for example calculating velocity fields from a video clip.

Each object supports a number of window-based **views** of itself. The views are available to the user, either through a popup menu selection, or by double clicking on the object. For each type of object a default view is defined, and it is this view that is displayed when the object is double clicked. As an example the video clip has four principal views, each of which provides different information to the user. The graph view is the default view. These views are discussed in detail in the accompanying **User's Guide**.

The user can choose to simultaneously display as many views of an object as they wish – even displaying the same view more than once. Whenever changes occur to an object each view updates its information automatically. When an object is closed – removed from the *FluidStream* system – all of its views are closed with it.

Each view has its own menu bar that provides the user with settings associated with that view. For instance, colours can be changed through the *Colour* menu of the video clip graph view.

This model of object, popup menu and views is consistent across all objects in the *FluidStream* system.

The *FluidStream* system itself is an object with four views. These views are displayed on opening the application and are described briefly below.

2.2.1 Application view

This window acts as the parent for all other windows in the application. In other words, it acts as a backdrop to all other windows and when other windows are minimised they appear at the bottom of the application view. When the application view is minimised all other windows are minimised with it.

This window contains nothing except the menu bar associated with the *FluidStream* application. There are seven menus, *Application*,

View, *Clip*, *Velocity*, *Density*, *Field* and *Info*. These are discussed in detail in the **User's Guide**.

This view is special in that closing it causes the application to exit.

2.2.2 Clip list view

A number of video clips can be open within *FluidStream* at the same time. Each video clip is listed in the *FluidStream* clip list view, and can be manipulated from this view.

2.2.3 Time series list view

A number of velocity or density field time series (see chapter 5) can be open within *FluidStream* at the same time. Each velocity or density field is listed in the *FluidStream* time series list view, and can be manipulated from this view.

2.2.4 Field list view

The user can create, and save to disk, a variety of fields derived from the velocity and density field time series. For example, the user can calculate the time averaged vorticity field from a velocity field time series. The field list view displays all field objects that have been opened in *FluidStream*.

2.3 Subsystems

FluidStream comprises three subsystems. These correspond to

- Image processing – particle identification
- Particle tracking, and
- Velocity field calculation.

The system does not provide support for direct image capture.

The particle identification subsystem will be discussed in chapter 3. The particle tracking will be discussed in chapter 4, and the velocity field calculation in chapter 5. In each chapter the objects associated with each subsystem will be introduced and discussed. Detailed information about the views available for these objects, and what

options are presented in their popup menus is given in the **User's Guide**.

2.4 The Java environment

FluidStream is written in the Java programming language. In theory this means that the program should be able to run on any computer platform provided a Java Runtime Environment (JRE, an implementation of the Java Virtual Machine (JVM)) is installed on the system. If your computer does not have a JVM installed then an appropriate JVM (this is platform dependent) can be downloaded from Sun's Java website. The address is

http://java.sun.com

Version 1.4 of the virtual machine is required.

Many computer systems have a JVM installed with their web browser to allow Java applets to be downloaded from the web and run on that computer. However some of the latest versions of *Windows* do not provide this.

FluidStream has been developed in a Windows environment, and tested extensively on computers running the *Windows 2000* and *Windows XP* operating systems. It has also been tested on *Apple Macintosh* systems running OS X.

2.4.1 Performance Issues

Java has not been traditionally used for computationally intensive software applications due to the fact that it was designed to be an interpreted language. In fact, early in its development, its performance in this area was known to lag well behind such languages as FORTRAN, C and C++. However, the introduction of JIT (Just In Time) compilers that produce native code, and thus avoid the speed limitations associated with interpreted code, has led to substantial improvements in computational performance.

The designers of *FluidStream* performed some simple tests in order to compare the computational performance of java with that of C++. A number of *FluidStream*'s core algorithms were rewritten in C++ and compiled, using *Microsoft Visual C++*, into *Windows* 32bit

applications. The results of this benchmarking suggested that the computational performance of the two languages was comparable.

2.4.2 Memory Management

The processing of large numbers of image files, and the manipulation of thousands, if not millions, of particles, places heavy demands on the memory and processing power of the computer. The systems currently used at the University of Canterbury have processors with clock speeds in excess of 2GHz, and a minimum of 1 Gb of RAM. Large, 120+ Gb, hard disks used solely for storing images and processed data, are also installed, and CD/DVD writers for archiving data are seen as essential to a viable laboratory system. Even with these relatively impressive specifications, by current standards at least, the system can still take considerable time (of the order of 10s of minutes) processing large video records (eg *1000*+ frames). Users of *FluidStream* who wish to handle densely seeded flows with long video records should try to access the highest performance machine possible for their work.

Due to this heavy demand on computer memory the system's memory management is of vital importance. When the system is first installed the user specifies the amount of RAM that is available to *FluidStream* for its storage and processing. See Appendix A for how this memory limit can be set.

The Java programming language handles memory in a particular way. Memory is dynamically allocated within the program when it is required, but recovery of this memory once it is no longer needed is beyond the programmer's control. Instead, Java has an automatic garbage collection mechanism that recovers memory automatically. While this relieves the programmer of considerable effort in memory management, it also leaves the responsibility of recovering memory with the Java Runtime Environment. If the runtime environment does not perform this task in a timely fashion it is possible for the application to become memory starved and to fail.

FluidStream has been designed to ask the runtime environment to recover memory when the available memory drops below a certain level. The user can set this level (see section 2.5.1). During memory, and processor, intensive activities, such as the particle matching process for a large video record, the user may be informed that the garbage collector is running, due to the fact that the available

memory has dropped below the preset level. This is normal behaviour, although, if it happens frequently during a particular process it may indicate that a RAM upgrade could be beneficial.

The user can ask the garbage collector to run manually by selecting *Run garbage collector* from the *FluidStream Application* menu.

The total amount of memory available to *FluidStream*, and the amount of memory currently unused, can be viewed by selecting *Memory* from the *Info* menu. This displays a dialog box that updates these statistics every second.

Occasionally the Java Runtime Environment fails to recover unused memory when it is required. This event generates an error that is displayed in the console window (see section 2.5.2) but it generally does not result in the application crashing. If this does occur you should save all data immediately and exit the program.

2.4.3 Versioning

As improvements to *FluidStream* are made new versions of the software are released. Data files created with older versions of *FluidStream* may not be compatible with the new version (this is due to the method by which Java stores data on disk). However *FluidStream* allows the user to save most data objects in a text format, that will be readable by future versions. You are encouraged to keep copies of each version of *FluidStream* after the release of each new version. A recommended strategy is to store the correct version of *FluidStream* with each set of files created by it. *FluidStream* data files can be substantial in size, depending on the length of the video record and the particle density. The additional memory required to save the *FluidStream* application files should be relatively unimportant.

2.5 Application basics

In this section a number of miscellaneous topics relating to the *FluidStream* application are discussed.

2.5.1 Configuration file

FluidStream creates a configuration file in the user's home directory when *FluidStream* is registered and run for the first time (see Appendix B). On subsequent occasions *FluidStream* will load this file on start-up. When the application exits the current configuration settings are saved to the configuration file.

The configuration file provides a number of user selectable parameters. These are

Current directory

Whenever the user tries to open a file, or save a new file, *FluidStream* will use the current directory as its starting point. Whenever the user changes the directory when opening or saving a file, the new directory is stored.

Minimum memory level

If the memory available to *FluidStream* drops below this level during certain memory intensive processes, then the garbage collector will be asked to run. This has a default value of *40Mb*.

Garbage collector wait time

When the garbage collector is requested to run the time allocated to it, before normal operations resume, is specified with this parameter. This has a default value of *1000ms*.

All of these parameters can be changed by selecting Properties from the Application menu. Also displayed in this dialog box is your Home directory. This parameter cannot be changed.

2.5.2 Console window

When the user performs an illegal operation a dialog box explaining the error is normally displayed. However, it is possible that some errors are not detected by *FluidStream* itself, but are caught by the runtime environment. This should be a rare occurrence, but when it happens an error message will be displayed in the black console window that is opened when *FluidStream* first starts. Examples of errors that will be displayed in the console window are out of

memory errors. In this case the user should save his/her data and close the application.

The console window will be open throughout a *FluidStream* session. Clicking the close button on this window will ungracefully exit the application. By this is meant that the user will not be prompted to save unsaved data. All such data will be lost. The console window should be minimised at start-up and not touched again unless the user realises that an error has occurred.

2.6 Summary

This section has provided a broad overview of some of the fundamental design concepts behind the *FluidStream* PTV system, as well as discussing issues associated with the Java programming environment.

3 Video clips

3.1 Overview of video clips

The fundamental object in the *FluidStream* system is the **video clip**. It represents a video record captured during an experiment. Such a record comprises a sequence of **video frames**, each of which corresponds to a single frame in the actual video record. Each video frame comprises a list of **particles** that have been identified in the particle identification process. Video clips can also be created from theoretical fluid flow models, but the basic concept of the video clip is unchanged. Only its source is different. The creation of clips, from both theoretical models, and sequences of digital images, will be covered in later sections.

The video clip, video frame and particle are all *FluidStream* objects. Full details of these objects can be found in the **User's Guide**.

In practice, two different image capture systems are used. In the first, video frames are captured at regular intervals, providing a stream of frames a fixed time step apart. In the second, the light source is pulsed and two frames are captured a short time apart. The time step between each pair of frames is considerably greater than that between the frames within a pair. The second system, referred to as a pulsed video clip, is generally used when the flow speeds are high. *FluidStream* supports both types of clip.

3.2 Video clip views

The video clip provides a range of views designed to give the user access to different aspects of the clip. These views are summarised in detail in the **User's Guide**. Here we will give an overview of the more important views.

3.2.1 Details view

The details view, like details views for all objects in the *FluidStream* system, simply displays fundamental information about the clip. Much of this information is entered by the user when the clip is created. It includes such things as the time step between frames, the

size of each frame in pixels and the scale factors used to convert from pixels to mm. A list of the frames, in numerical order, is also displayed, allowing the user to interact with the frames within the clip directly through their popup menus.

Information about the creation of the clip, whether from a sequence of images or from a theoretical model, is also available through this view.

3.2.2 Graph view

The graph view is by far the most useful, and because of this it is the default view for a video clip.

The graph view provides a TV screen simulation of the clip. Frames within the clip can be viewed individually or together. The clip can be traversed forward or backward one frame at a time, or reviewed as an animated sequence in real time, slow motion or fast forward. Viewing several frames at the same time can provide the user with significant insight into the flow structure. In many ways this feature simulates a long exposure photograph that clearly displays fluid pathlines in the flow.

Information from the matching process can be superimposed on the particles themselves.

The user can extract additional information about the matching process by selecting regions within the graph view. This capability is discussed further in the **User's Guide**.

The graph view also provides the user with a way to remove extraneous particles from a flow and delete undesirable matches. The processing of real images may sometimes identify particles that don't actually exist within the flow. These normally arise due to light reflections off solid boundaries or a free surface, and their presence can adversely affect the matching process.

3.2.3 Match summary view

The match summary view provides information about the results of the matching process. A table, containing one row for each frame, displays the number of particles in the frame and the number of PTV matches that were made. For model-generated clips a range of other

statistics appear in additional columns. These statistics are associated with the number of correct, incorrect, missed, and false matches. This information is vital if a particular matching strategy is to be evaluated.

The match summary view also provides an analysis of the particle path lengths generated by the matching process. The length of a particle path is defined to be the number of particles within it. Thus a particle that is left unmatched has a path length of 1, and a particle that can be traced through every frame in a clip will have a path length of *N*, where *N* is the number of frames in the clip.

3.2.4 Analysis view

The analysis view allows the user to manage the PTV analyses defined for the clip. The user is at liberty to define as many analyses as they wish. These analyses execute sequentially.

The analysis view provides the user with control over these analyses. They can create new analyses to add to the ones already defined, they can load analyses from disk, they can delete analyses, they can edit them, they can reset them so that they can run again, they can clear all analyses from the clip, and finally they can execute them.

The user has considerable flexibility here. After one analysis has been executed they can decide to create and execute a new one. Or they may wish to add two new analyses before executing them. In each case the new analyses work with the matches made by any previous analyses. The PTV analysis object, and its component parts will be discussed at length in chapter 4 and further information is available in the **User's Guide**.

3.3 Video clip creation from image sequences

To begin we consider the creation of a video clip from a sequence of experimentally obtained digital images.

This process involves the identification of particles within each image using an appropriate image-processing algorithm. Three algorithms are provided by *FluidStream* and these are discussed in detail below. Each algorithm assumes that the particles are

illuminated by a light sheet, in an otherwise dark environment. Thus the particles appear as small regions of pixels with high intensity embedded in a dark background. For some applications this may not be the case. For instance, in surface drogue tracking experiments the drogues may be dark coloured objects illuminated by ambient light. In this case the sister application, *ImageStream*, can be used to pre-process the video images in order to convert them into the desired form. *ImageStream* is also part of the *Streams* software library.

3.3.1 File Formats

FluidStream doesn't deal directly with digital videos. Instead, it works with sequences of digital images stored as JPEG, PNG, BMP, TIF or GIF files. Such sequences may arise from a number of sources including digital still, and video, cameras. In our laboratory digital videos are recorded on computer using *Adobe Premiere 6.0* or custom designed *Labview* VIs. These software packages can export a digital video as a sequence of TIF, GIF, JPEG or PNG files, which are read directly by *FluidStream* (or *ImageStream*).

It is worth noting the differences between the different file formats. The JPEG format is compressed and lossy while the PNG, BMP, TIF and GIF formats are lossless. The GIF format, however, has a colour palette limited to 256 colours. For particle tracking applications the slight noise introduced into the JPEG files, and the lack of complete colour information in the GIF files, generally is not important. On the other hand the file sizes tend to be significantly different between the different formats, with JPEG tending to be the smallest and TIF and BMP the largest.

3.3.2 Creation process

Video clips can be created from sequences of image files by selecting *Create clip from image sequence* from the *FluidStream Clip* menu, and choosing one of the three algorithms available (see section 3.3.3). The user is presented with a dialog box in which to enter relevant information for the creation process (see the **User's Guide**).

The user selects the files to be analysed and records the space scales, time increment for the clip, and whether the clip is pulsed (if it is then the pulsed time step is also required). It is also possible to limit the analysis to a sub-window of each frame. This can be useful if

there is some region of each image that contains light signatures that might be incorrectly mistaken for particles.

Each of the particle identification algorithms has a number of parameters that must be set by the user – the default values are unlikely to be optimal. The user is provided with a means of exploring the effect of changing these parameters by opening an *Image analysis* window that displays the image of a particular frame as well as the particles that are identified with certain parameters settings. This window also lists details of the identified particles. See **User's Guide** for more information on this window.

3.3.3 Particle identification algorithm

Particle identification from digital images is not a trivial process. A range of algorithms can be devised to do this, each with its own strengths and weaknesses. Each image to be analysed is converted into a two-dimensional array of pixel colour values. The colour value has red, green and blue intensities stored as *8* bit integers (ie they lie in the range *0-255*). A particle identification algorithm explores this pixel array identifying particles based on certain criteria. Different algorithms may identify different particles in the same image, and, in fact, the same algorithm will identify different particles if its operating parameters are changed.

Currently *FluidStream* has three particle identification algorithms implemented (note that the second and third of these are based on the same general concepts).These are

- the average algorithm, and
- the Gaussian absolute algorithm
- the Gaussian relative algorithm

and they are described in the following sections.

3.3.3.1 Average algorithm

The average algorithm searches the pixel array for pixels whose intensities exceed a certain threshold value, defined by the user. The intensity can be based on any of the individual colour values (red, green or blue), or on an average of all three.

Once a pixel, with an intensity greater than the threshold, is found, the pixels surrounding this pixel are traversed so that the particle's physical extent can be determined. All pixels with intensities greater than the threshold, and adjacent to another pixel within the particle, are identified as part of that particle.

For each particle identified in this way the size, intensity and physical location are recorded. The position is calculated from the centre of mass of the pixels within the particle. The size of the particle, represented by a radius, is determined from the number of pixels that comprise the particle, based on the assumption that the particle is circular. Its intensity is the average intensity of the pixels making up the particle.

In addition to specifying the threshold intensity the user can invoke limits on the particle's minimum and maximum size. The minimum particle size limit can be useful for eliminating very small regions that do not correspond to actual particles.

One drawback of this algorithm is the limited accuracy with which the particle's location can determined. The location cannot be resolved to better than half a pixel and therefore the location is accurate to a quarter of a pixel.

3.3.3.2 Gaussian algorithms

The two Gaussian algorithms use a rather more sophisticated algorithm for particle identification and, in particular, for the calculation of the particle location. In this case variations of intensity within a particle are used to determine the particle's location to sub-pixel accuracy.

These algorithms start by searching for a pixel whose intensity is a local maximum and exceeds a maximum threshold as set by the user. Adjacent pixels that exceed a second threshold value are assumed to be part of the same particle provided no other local intensity maximum occurs within the region, in which case a second particle is recorded.

A Gaussian intensity distribution is assumed for each particle, and a Gaussian curve is fitted to the maximum intensity and the intensities of the pixels on either side of it, in both the *x* and *y* directions. In this way sub-pixel accuracy is obtained for the particle position (see

Cowen and Monismith 1997) provided the particle extends over at least three pixels in each direction. The accuracy of the locations of particles smaller than this is the same as for the average algorithm.

The radius of each particle is calculated from the number of pixels either side of the pixel (with maximum intensity) that exceed the lower threshold value. As for the average algorithm the user can select which colour values are used in calculating the intensity of a pixel.

There are two Gaussian based algorithms. The first, the Gaussian absolute algorithm, uses absolute intensities in determining the threshold values. These absolute intensities can be measured relative to the local background intensity, or to an intensity of zero. The former is useful when the lightsheet intensity is not uniform across the image. The background intensity is calculated by averaging the pixel intensity in a small region surrounding the pixel of interest. The size of this area can be specified by the user. The processing speed is substantially affected by the use of the local background intensity, so should not be used unless it is deemed necessary.

The second Gaussian based algorithm is the Gaussian relative algorithm. This works in the same way as the absolute algorithm except all thresholds are based on the ratio of the actual intensity to the local background intensity. Thus intensity factors are specified for the threshold values. As this algorithm always uses the local background intensity in its calculations it is relatively slow in processing the image files.

3.3.4 Guidelines

In this section we provide some general guidelines for creating video clips from image sequences.

- as a rough guide, the maximum number of particles per frame should not exceed 2000-3000. Larger numbers of particles will tend to adversely affect *FluidStream*'s computational performance. The number of particles is controlled by the threshold settings for both algorithms.
- the user should always look at the clip's graph view once it has been created. By displaying a number of the frames at once the user can obtain a good sense of the validity of the particle identification process. If the view shows no clear flow structure the creation process may need to be reviewed.

- the user should check the match summary view for the clip to see if the number of particles per frame is roughly the same for all frames. If this is not the case, then the data is likely to be of low quality unless there is a good physical reason for the observed variations.
- if possible one of the Gaussian image algorithms for particle identification should be used.

3.4 Video clip creation from theoretical models

Theoretically generated clips can be effectively used to evaluate different particle matching algorithms, and have been responsible for driving much of the analytical development within the *FluidStream* system. New users of *FluidStream* will find that the theoretical flows provide a powerful environment in which to learn the system's capabilities and limitations. Experimentation with these theoretical models is strongly encouraged.

For each model a set of randomly located particles is generated in the first frame in the clip. Using the known velocity fields, the displacement of each particle is calculated from frame to frame using a 4^{th} order Runge-Kutta time-integration of the ordinary differential equations governing particle motion. A particle may leave a frame by flowing through one of its edges, and the user has the choice of reseeding such particles through the opposite edge. The loss of particles from the light sheet can be simulated by randomly removing, and reseeding, particles in each frame.

While all of the theoretical flows are laminar in nature (although some contain features that attempt to model aspects of a turbulent flow), many of the practical issues affecting the performance of a PTV system, such as velocity shear and particle disappearance, are simulated, and allow realistic exploration of system performance in experimentally measured flows.

Four theoretical models are available and these are discussed in the following sections. Full details for creating clips from theoretical flow models can be found in the **User's Guide**.

3.4.1 Cellular model

This model creates a laminar cellular flow specified by a streamfunction of the following form

$$\psi = A \sin\left(\frac{n\pi x}{w}\right) \sin\left(\frac{m\pi y}{h}\right) \tag{3.1}$$

Here *w* and *h* are the width and height of the flow in mm. *A*, *n* and *m* are constants set by the user – note that *m* and *n* are integer values, and *A* is in mm^2/s. The frame edge is a streamline so no particles flow out of the frame.

3.4.2 Irregular cellular model

This second model is a somewhat more complex version of the cellular model. Two different cells are superimposed on a uniform flow in the *x* direction. In addition the cells have an oscillatory time dependence.

$$\begin{aligned}\psi = {} & A_1 \sin\left(\frac{n_1\pi x}{w}\right) \sin\left(\frac{m_1\pi y}{h}\right) \cos\left(\omega_1 t\right) \\ & + A_2 \sin\left(\frac{n_2\pi x}{w}\right) \sin\left(\frac{m_2\pi y}{h}\right) \cos\left(\omega_2 t\right) + Uy\end{aligned} \tag{3.2}$$

Here *w* and *h* are the width and height of the flow in mm. A_1, n_1, m_1, ω_1, A_2, n_2, m_2, ω_2 and *U* are constants set by the user – note that n_1, m_1, n_2 and m_2 are not necessarily integer values.

3.4.3 Multi-cellular model

This flow attempts to create an artificial flow including a range of length scales, each with an associated velocity scale. It is a crude attempt to produce a flow with homogeneous, mean shear free turbulent-like characteristics.

The flow is composed of a number of cell flows generated using the streamfunction of equation 3.3.

$$\psi = A \sin\left(\frac{n\pi x}{w}\right) \sin\left(\frac{m\pi y}{h}\right) \cos\left(\omega t\right) \tag{3.3}$$

The user specifies the constants A, n, m and ω for the largest cells, the number of cell sizes in the flow, N, and the ratio of the smallest to largest cell sizes, L_r. The model automatically generates the constants for the other cell sizes based on the length ratio and those for the large cells. The formulae used for the i^{th} cell size (*1* being the smallest) are

$$\begin{aligned} z_i &= \left(L_r\right)^{N-i/N-1} \\ A_i &= A\left(z_i\right)^{4/3} \\ n_i &= \frac{n}{z_i} \\ m_i &= \frac{m}{z_i} \\ \omega_i &= \omega\left(z_i\right)^{-2/3} \end{aligned} \tag{3.4}$$

The streamfunctions for the N cells are superimposed to produce the flow.

3.4.4 Turbulent shear model

The final theoretical model creates a flow with mean shear together with superimposed velocity fluctuations. The velocity field is chosen to ensure that continuity is satisfied.

$$u = U_{mean} + U_{top}\left(\frac{y}{h}\right)^{1/n} + u'\sin(\omega t)\sin\left(\frac{ky}{h}\right)\exp\left(\frac{-ry}{h}\right) \quad (3.5)$$

$$v = -v'\sin(\omega t) \quad (3.6)$$

Here h is the frame height in mm. U_{mean}, U_{top}, n, u', k, r, v' and ω are constants set by the user.

3.5 Opening and saving video clip files

Video clips can be saved to disk, in an object or text file, through a popup menu choice. All information relating to the clip is saved with it. Video clips, that have previously been saved, can be opened from the *FluidStream Clip* menu. All video clips stored as objects have a file extension of *.clp7*, while text files have a file extension of *.ctx7*.

3.6 Summary

This chapter has introduced the video clip, has covered its views, and discussed the creation of video clips from both sequences of images and theoretical flow models. The advantages of using the theoretical models as a vehicle for exploring *FluidStream*'s performance have been noted.

The three algorithms provided with *FluidStream* for particle identification in digital images have been presented. These algorithms work with sequences of digital images. The importance of the particle identification process cannot be over-emphasised, as the success of the PTV analysis process is dependent on good quality particle frames.

4 Particle Tracking

4.1 Introduction

The core of the *FluidStream* system is its ability to match particles from frame to frame in a video clip. Without this ability the information obtained from the video record is qualitative in nature, providing only a broad-brush picture of the flow structures and their time dependence.

In this section we will look at the particle matching process in some detail. This process will be called by a number of synonymous names, such as PTV analysis, particle matching process or procedure.

4.2 Optimisation process

To begin we will consider the theoretical problem of matching particles in one frame to those in another. Later we will talk specifically about *FluidStream*'s implementation of this process.

The particle matching problem can be considered as a general optimisation problem, known as the **assignment problem**. In the assignment problem two groups of objects are defined. The aim of the problem is to assign objects in the first of these groups to those in the second based on costs associated with each assignment. The optimal solution is obtained when the total cost of the assignments is minimised.

Figure 5.1 illustrates a simple example of the assignment problem and its optimal solution.

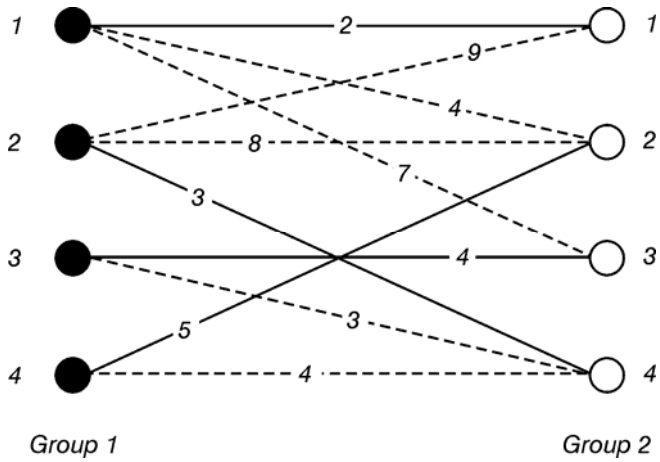

Figure 4.1 A simple example of an assignment problem. There are four objects in each of the groups. The lines indicate the possible assignments and the associated costs, c_{ij}, associated with assigning object *i* from group *1* to object *j* from group *2*. The optimal solution is obtained when the total allocation cost is minimal. For this problem the optimal assignments are illustrated with bold lines.

The particle-matching problem is clearly of the assignment type. Particles in the first frame comprise the first group of objects, while those in the second comprise the second group. Costs need to be assigned to each of the prospective matches between a frame one particle and a frame two particle. This can be done in a multitude of ways, and we will discuss a number of such costing strategies, as implemented in *FluidStream*, in section 4.4.

Even so, the particle-matching problem does not fit the assignment model exactly. The standard assignment problem requires the same number of objects in each of the two groups, so that each object in the first group is assigned to an object in the second group, and that the cost associated with each assignment is integral.

The number of particles present in each of two subsequent frames is nearly always different, and integral costs are unlikely to occur. Therefore the problem is a somewhat modified assignment problem.

A number of solution techniques are available for solving the assignment problem. The most general of these is a linear programming technique known as the **simplex method**, although it is known to be computationally inefficient. However, because of its special structure (the assignment problem is a very specific linear programming problem) more specialised techniques have been developed. The currently favoured technique is known as the **auction algorithm**. This algorithm can be adapted to the modified assignment problem that corresponds to the particle-matching problem, and this be discussed in broad terms in the next section.

4.2.1 The auction algorithm

The auction algorithm, in the context of the particle-matching process, can be understood in the following simple terms (see Bertsekas 1992). The particles in the first frame are referred to as bidders in an auction, where the particles in the second frame are the objects for which the bidders are bidding. Each object carries with it a benefit for some of the bidders in the auction. A high benefit, corresponding to a low cost, means that a bidder desires the object highly. This benefit is defined to be the maximum matching cost (MMC) less the cost, where the MMC is defined in the next section.

The objective of the auction algorithm is to allocate objects to bidders so that the overall benefit accrued to the bidders is maximised. This is equivalent to allocating particles in frame 2 to particles in frame 1 such that the overall cost is minimised – which is the aim of the particle-matching algorithm.

Initially, all bidders are placed in a pool of unallocated bidders, and the auction is ready to start. All objects start with a price of *0*, but as bidders bid for them, their prices rise. Unallocated bidders bid for objects if they can find one for which the net benefit for them is greater than *0*. The net benefit is defined as the object's benefit less the object's current price. In *FluidStream* each unallocated bidder is considered in turn. If an object with a positive net benefit exists for that bidder then the bidder bids for the object with maximum net benefit. The bid increment is based on the difference between the net benefit of the most desirable object and the net benefit of the next most desirable object. This bid increment is added to the price of the most desirable object and this object is allocated to the current bidder. If this object was previously allocated to another bidder this bidder returns to the unallocated pool.

If a bidder in the unallocated pool cannot find an object that has a positive net benefit then they drop out of the auction and remain without an object. This corresponds to a particle in frame 1 being left unmatched.

The algorithm sweeps through the list of unallocated bidders repeatedly until all unallocated bidders have dropped out of the auction, or they have been allocated an object.

The algorithm just described is called the **naïve auction algorithm** because it may fail to complete. The conditions for this happening are easily seen. If a number of bidders have two or more objects with the same positive net benefit, then the naïve auction algorithm requires that the bid increment must be *0*. Therefore two or more bidders can repeatedly swap objects without the prices for these objects increasing. A modified auction algorithm can be developed to overcome this circumstance although it is not clear how it can be implemented in the particle tracking problem where the cost (and hence the benefit) associated with an object can take on any real value.

In *FluidStream* a modified naïve auction algorithm is implemented. To avoid non-convergence the user specifies the maximum number of sweeps that are made through the list of unallocated bidders (the default is *1000*). It has been found that when meaningful costings are used, convergence is not an issue. In other cases, only a few particles are affected, and the final allocations of these are generally acceptable.

As the optimisation process proceeds the user is informed of the number of bids made during the auction as well the number of sweeps taken, and the number of bidders that were left unallocated.

4.3 PTV analysis

Before discussing the specific costing strategies implemented in *FluidStream* we will introduce an overview of the particle matching process.

All of the information needed to match particles in a video clip is defined in a **PTV analysis** object. The user can create a PTV analysis object through the analysis view of the clip. This view provides mechanisms for the creation, deletion, editing, resetting,

clearance and execution of PTV analysis objects. The user is free to define as many of these analysis objects for a clip as he or she wishes.

A PTV analysis object works through the sequence of frames in the clip in pairs, attempting to calculate optimal matches for the particles in the first of the frames based on the specified costings. Note that for a pulsed clip *FluidStream* only works with the pairs of frames that are temporally close together. No attempt is made to match particles between frames in different pairs. The user can choose to limit the analysis to a subset of the frames if they wish. In addition, the user can instruct the analysis object to start the analysis at the first frame and continue forward to the last frame, or alternatively to start the analysis at the last frame and work backwards to the first. The analysis in each direction is the same, except of course, for the fact that in the second circumstance time has been reversed.

While the particle matching process between two frames is a linear programming problem, the overall problem, matching particles in all frames in a clip, may be non-linear. Some of the costing strategies discussed in section 4.4 rely on matches in subsequent or previous frames, and therefore, as the analysis continues, and matches change due to the optimisation process, previously allocated matches may become non-optimal. To alleviate this problem somewhat the user can instruct the optimisation process to iterate through the frames a number of times. In doing this it is assumed that the solution will converge towards a global optimal solution. We provide no proofs for whether this is the case or not, and it is quite possible that for some problems such an iterative technique does not converge to the desired solution. However, handling the full non-linear problem is computationally too expensive to implement.

When more than one iteration through the frames is selected the user may choose to perform each iteration in the opposite direction to the one before.

Now let us consider the details of the optimisation process at the level of an individual frame. In the most general case, every particle in the first frame could be potentially matched to any particle in the second. This level of generality is computationally prohibitive. Consider a clip where each frame contains around *2000* particles (this is not unusual in practice). The number of possible matches is the product of the number of particles in each frame, giving rise, in

this case, to potentially *4* million matches. Each of these matches must be costed. For some costing strategies, each cost may require tens of thousands of calculations. The end result is that simply calculating the costs for the particles in one frame may require tens of billions of calculations. Without a supercomputer this requirement is crippling.

To circumvent this problem a **search window** is defined. The search window is a rectangular region defined relative to a particle in the first frame of the two being considered. Only particles lying within this physical rectangle in the second frame are considered as possible matches to the particle in the first frame. By a judicious use of the search rectangle the number of candidate matches for each frame 1 particle may be reduced to a fraction of the total number of particles in the second frame. The search window can be defined so that the particle in the first frame does not even lie within it. This may be useful for flows that are essentially uni-directional.

Once the set of possible candidate matches has been deduced for each particle in the first frame the analysis process calculates the costs of these matches using the costing strategies selected by the user. Using these costs the set of possible matches for each particle is pruned further. Each PTV analysis object defines a **maximum matching cost** (MMC), set by the user. This cost serves as a cut-off between costs that are viewed as reasonable, and costs that are seen as unreasonable. Any match that has a cost exceeding the MMC is eliminated as being undesirable. While the search window is a crude method of reducing the number of possible candidate matches for a particle, the use of the MMC is a rather more refined method of achieving the same objective. Its setting is crucial to a successful analysis. If the MMC is set too low, then correct matches will be excluded, and at best these matches will be missed. If the MMC is set too large too many spurious candidate matches will be considered, costing computational time, but also allowing incorrect matches to be made for particles that perhaps have no correct match in the next frame. The selection of an appropriate MMC is a skill that comes with experience, although experimentation with the theoretical model flows is an effective way of evaluating various choices of MMC for different costing strategies.

FluidStream provides another technique for controlling the matching process. The user can define **regions** in the video clip graph view. These regions can be rectangular, circular, polygonal or freehand-

drawn. Each PTV analysis object can define a number of regions, for both the first and second frame, that can be used to exclude particles contained within those regions from the matching process.

Once the candidate matches, whose costs exceed the MMC, have been eliminated a final set of candidate matches for each particle in the first frame is available, and the auction algorithm, as discussed in section 4.2.1, can be executed for the pair of frames under consideration. Once this optimisation process is complete, the analysis moves to the next pair of frames, the first of which is the second frame from the previous pair. And so the process continues until all frames have been analysed.

As the matching process executes a **PTV analysis monitor dialog box** is displayed, keeping the user informed of progress. The initiation and completion of each step in the process is logged, together with the time taken for it to execute. For the optimisation process details of the auction algorithm are recorded.

As has been alluded to earlier the computational requirement of this process can be prohibitive. Therefore it is essential that the user make sensible parameter selections in order to reduce the computational demand. At this stage the most important parameter in this regard is the search window. The smaller this window is the better.

4.4 Costings

Each costing employs a particular algorithm for calculating the cost of a match between two candidate particles. This algorithm usually involves a number of parameters that can be set by the user, and the performance of the costing is strongly dependent on the parameter choices.

All costs in *FluidStream* are non-dimensional. This eliminates the problem of combining costs with different dimensions. In addition, each cost is non-dimensionalised in such a way that, if possible, it lies between *0* and *1*. A cost of *0* corresponds to a highly desirable match, while a cost of *1* corresponds to the least desirable match. For some costings the cost can be guaranteed to lie with this range. For others this is not possible. The MMC will normally be selected to lie between *0* and *1* when a single costing strategy is used.

The key to generating the correct particle matches between frames is the generation of accurate costs. Clearly, if the correct match for every particle has the smallest cost of all the possible matches, and particles that have no correct match in the next frame, have no candidate matches in the next frame, then the optimisation process will generate a perfect set of matches for every frame.

These fortuitous circumstances are, unfortunately, unlikely to occur in practice, and it is through the skill of the user that satisfactory sets of matches are achieved.

Here we will consider the components that affect the success of the PTV analysis and discuss the roles they play in the process.

Costing strategy

The role of a costing strategy is to generate costs in such a way that the correct match carries a cost substantially less than all incorrect matches. In general, this cannot be achieved for all particles in a frame, in which case the costing must at least try to avoid an incorrect match having a substantially lower cost than the correct one (in this way the optimisation process may still select the correct match as the other desirable incorrect matches may be correctly matched to other particles). If this cannot be achieved then the costing may be a poor choice for the particular flow. Note, most costings have a range of parameters that affect their performance, and therefore a costing with one set of parameters may perform poorly, while the same costing with another set of parameters may perform well.

The costing also has the duty of ensuring that when a particle has no correct match in the next frame, no incorrect match is given a low cost. In fact it is most desirous that all incorrect matches have costs exceeding the MMC.

MMC

The role of the MMC is to eliminate all incorrect matches from consideration, leaving only correct matches as candidate matches. If the costing strategy is able to clearly delineate the correct and incorrect matches then such a goal may be achieved. In practice the MMC is likely to be used to eliminate most of the incorrect matches,

and in the case where the correct match has a similar cost to some of the incorrect matches, it needs to be carefully chosen so as to not exclude the correct match altogether. When no correct match exists the role of the MMC is to eliminate all candidate matches.

Default Cost

Some costing strategies cannot calculate a cost in certain circumstances. A simple example is the recent velocity costing. This costing needs matching information from the previous, or next, frame in order to calculate its cost. If such information is not available then no cost can be evaluated. In such circumstances the costing must specify a **default cost** that will be used when an actual cost is unavailable. This default cost should be greater than the MMC so that a match with the default cost is never considered. The philosophy here is that no news is bad news. A particle that can't be identified as the correct match, or an incorrect match, must be treated as an incorrect one. The reason for this is that incorrect matches tend to degrade the computed velocity field considerably more than missed matches.

Our discussion up until this point has focussed on the idea that only one costing strategy is used at a time. A more general approach is to use several costings simultaneously. This adds to the complexity of the analysis considerably.

The first consideration is how the costs generated by each of the costings contribute to the overall cost. This is handled by allocating each costing a **weighting**. The calculated cost for each costing is multiplied by its weight, and the resulting costs are added together. Thus the user is at liberty to impose an order of importance on the chosen costing strategies.

If all costings chosen specify a default cost then it is important that the weighted sum of the default costs should exceed the MMC. However the choice of MMC, default costs and weightings becomes more challenging when some of the costings have defaults and others do not.

Costings can also be combined in another way by specifying one or more costings as **default costings**. This is most clearly illustrated through an example. Suppose a PTV analysis object includes two costings, say A and B, and costing B is designated as a default costing. When the cost of a particular match is computed initially,

only costing A is used. However, if costing A is unable to compute a cost, instead of using its default cost, costing B is used to compute the cost. In this way costing B is used only as a fallback position, with costing A playing the dominant role in the matching process.

The experience of *FluidStream*'s designers is that a number of PTV analyses, each with a single, but different, costing is rather more manageable than a single PTV analysis with multiple costings. However, this conclusion needs further testing.

4.4.1 Costing Types

A total of 15 different costings are available in *FluidStream*. These costings can be classified into two groups based on whether they require matches between particles in order to calculate their cost.

State based costings only require information on the state of each particle in a frame. Such information includes the particle's location, size and intensity. **Matching based costings** use matches between particles to determine their costs. Therefore only state based costings can be used to initiate the particle tracking process, or at least, if multiple costings are to be used, then at least one of these must be state based. Once an initial set of matches has been calculated, matching based costings may be used.

Another important difference between the two costing types is that the cost associated with a particular match is invariant for a state-based strategy, while for a matching-based strategy the cost will depend on the current set of matches. For this reason the performance of a matching-based strategy may improve if it is iteratively applied to the frame sequence.

In general, the matching based costings are more flexible than their state based cousins. This is due to the fact that they can use dynamic information such as particle velocities to help the matching process.

In the next few sections each of the costing strategies will be briefly reviewed. Full details of each costing can be found in the **User's Guide**.

In the discussions that follow we will refer to the two frames being analysed as frames 1 and 2, frame 1 being the frame whose particles

are being matched. P1 will be used to refer to an arbitrary particle in frame 1, and p2 to refer to an arbitrary particle in frame 2.

4.4.2 State based Costings

There are six state based costings. These are

- adjacency costing (see section 4.4.2.1)
- centre of mass costing (see section 4.4.2.2)
- correlation costing (see section 4.4.2.3)
- distance costing (see section 4.4.2.4)
- intensity costing (see section 4.4.2.5) and
- size costing (see section 4.4.2.6)

4.4.2.1 Adjacency costing

The **adjacency costing** is designed to measure the degree to which the pattern of particles surrounding particle p1 corresponds to the same pattern surrounding p2. It has much in common with the correlation costing but is computationally considerably less expensive.

The user specifies a rectangular region surrounding particle p1 and all particles lying within that region are found. The same is done for particle p2. All particles in the p1 rectangle are "displaced" by the vector $\mathbf{r}_{p2} - \mathbf{r}_{p1}$, where $\mathbf{r}_{p2}$ is the position of particle p2 and $\mathbf{r}_{p1}$ is the position of p1. The list of particles in the p2 rectangle is then searched for the particle that is closest (adjacent) to each of the displaced frame 1 particles. The costing is the sum of the distances between these closest matches normalised by the distance between p1 and p2. Thus

$$C_{12} = \frac{\sum_{i=1}^{n_1} \left| \mathbf{r}_i + \mathbf{r}_{p2} - \mathbf{r}_{p1} - \mathbf{r}_{\alpha_i} \right|}{n_1 \left| \mathbf{r}_{p2} - \mathbf{r}_{p1} \right|} \qquad (4.1)$$

- n_1 – the number of particles in the frame 1 rectangle that have closest neighbours in the frame 2 rectangle.
- $\mathbf{r}_i$ – the position of particle i in the frame 1 rectangle
- $\mathbf{r}_{\alpha i}$ – the position of the particle in the frame 2 rectangle that is closest to particle i in frame 1.

The user has a number of ways to customise this costing. The contribution of each frame 1 particle to the total cost may be weighted by the relative distance the particle is from p1. Closer particles have a stronger influence on the total cost. A limit can be placed on the maximum distance used in the cost calculation, and the user can choose whether each particle in frame 1 must have a unique partner in frame 2.

4.4.2.2 Centre of mass costing

The **centre of mass costing**, like the adjacency costing, attempts to compare particle patterns. It does this in a gross way by calculating the positions of the centres of mass of the particles in the rectangles centred on p1 and p2. In calculating the centre of mass the actual size of the particles is not taken into account – in other words all particles are assumed to be the same size. The distance between the centres of mass, normalised by the distance between p1 and p2 provides the cost. Thus

$$C_{12} = \frac{\left|\mathbf{r}_{cm1} - \mathbf{r}_{cm2}\right|}{\left|\mathbf{r}_{p2} - \mathbf{r}_{p1}\right|} \qquad (4.2)$$

- $\mathbf{r}_{cm1}$ – the position of the centre of mass of the particles surrounding p1
- $\mathbf{r}_{cm2}$ – the position of the centre of mass of the particles surrounding p2.

4.4.2.3 Correlation costing

The **correlation costing** is based on the cross-correlation of the intensity field of sub-windows in the two frames. In this sense it is similar to the correlation used in a PIV system.

The user chooses a rectangular sub-window centred on p1 and overlays this sub-window on frame 2, centring it on p2. The cross-correlation of the intensities in the two sub-windows is calculated, and normalised by the intensity auto-correlations of the two sub-windows. This calculation provides a measure of the similarity of the particle patterns in the two frames, and hence how likely particles p1 and p2 are the same particle.

Thus the cost can be expressed as

$$C_{12} = 1 - \frac{cross\,correlation_{12}}{\sqrt{auto_1 auto_2}} \qquad (4.3)$$

- $cross\ correlation_{12}$ – the cross correlation of intensities in the two rectangles excluding particles p1 and p2
- $auto_1$ – the auto-correlation of the frame 1 rectangle excluding p1
- $auto_2$ – the auto-correlation of the frame 2 rectangle excluding p2

This process differs from the PIV calculation in a number of ways. Firstly the calculation is performed to determine the "goodness" of the match between p1 and p2. It is not used to find the displacement of the sub-window that leads to the highest correlation. Secondly, particles p1 and p2 are excluded from the correlation calculation. A near perfect correlation between these two particles is guaranteed by the choice of sub-window displacement, and it is the correlation of the particles around them that determines whether they are the same particle. Thirdly, as the correlation is calculated using artificially created frames of pixels (unlike the PIV system that uses the original digital images) a variety of manipulations can be performed on the particles in the artificial frame. The most important of these is the enlargement or compression of the particles in the frame. This capability is found to provide a powerful enhancement to the performance of the correlation costing.

4.4.2.4 Distance costing

The **distance costing** is perhaps the simplest, and most easily understood, costing. It is based on the distance separating the two particles for which the cost is being calculated. In its original form the cost was proportional to the distance separating the two particles, normalized by half of the distance from one corner of the search

window to the corner diagonally opposite. However, this definition has been generalized in a number of ways. Firstly, the costing may be based on the distance between the particles, or its square, thus more severely penalizing matches between more distant particles. Secondly, the user is at liberty to suggest how far particles move between frames so that the distance is computed from the difference between the actual separation and that set by the user. Here too, the user has a choice. For a uni-directional flow where all particles are expected to move in a particular direction the user can specify the expected displacement of a particle between frames both in terms of its magnitude and direction. Alternatively only the magnitude of the distance can be specified. If this distance is zero then the costing resembles the original definition. The cost of a match between p1 and p2 is given by

$$C_{12} = \frac{distance}{1/2\sqrt{sw_x^{\,2} + sw_y^{\,2}}} \qquad (4.4)$$

- *distance* can be defined in a number of ways. If the user suggests no displacement between frames then *distance* = $\left|\mathbf{r}_{p_2} - \mathbf{r}_{p_1}\right|$. If the magnitude of the likely displacement, $\left|\Delta\mathbf{r}\right|$, is provided then distance = $\left|\left|\mathbf{r}_{p_2} - \mathbf{r}_{p_1}\right| - \left|\Delta\mathbf{r}\right|\right|$, and if both the magnitude and direction are specified, then *distance* = $\left|\mathbf{r}_{p_2} - \mathbf{r}_{p_1} - \Delta\mathbf{r}\right|$.
- $\mathbf{r}_{p1}$ – the position vector of p1
- $\mathbf{r}_{p2}$ – the position vector of p2
- sw_x – the width of the PTV analysis search window
- sw_y – the height of the PTV analysis search window

If the user chooses to use the square of the distance then the cost equals $C_{12}^{\,2}$.

4.4.2.5 Intensity costing

The **intensity costing** measures the change in intensity as a particle moves. It assumes that the intensity of a particle should change little between frames.

The cost is calculated by

$$C_{12} = \frac{|I_1 - I_2|}{255} \tag{4.5}$$

- I_1 – the intensity of p1 (on an *8*-bit scale of *0-255*)
- I_2 – the intensity of p2 (on an *8*-bit scale of *0-255*)

4.4.2.6 Size costing

The **size costing** is based on the change in particle size from frame to frame. It assumes that the particle's size should remain constant, or nearly constant, as it moves.

The cost is calculated by

$$C_{12} = \frac{|r_1 - r_2|}{\max\{r_1, r_2\}} \tag{4.6}$$

- r_1 – the radius of p1
- r_2 – the radius of p2

4.4.3 Matching based Costings

Matching based costings rely on particle matches between frames to calculate their costs. Unlike state based costings they can take into account information about the flow itself, and, for this reason, unlike state based costings, they cannot be used to start the matching process. This reliance on particle matches means that matching based costings may benefit from multiple iterations of the optimisation process.

There are nine matching based costings. These are

- least squares velocity costing (see section 4.4.3.1)
- local velocity costing (see section 4.4.3.2)
- magnitude velocity costing (see section 4.4.3.3)
- path length costing (see section 4.4.3.4)
- polynomial velocity costing (see section 4.4.3.5)

- recent acceleration costing (see section 4.4.3.6)
- recent velocity costing (see section 4.4.3.7)
- space average acceleration costing (see section 4.4.3.8) and
- time average velocity costing (see section 4.4.3.9)

All of these costings, except the path length costing, require a default cost factor.

4.4.3.1 Least squares velocity costing

The **least squares velocity costing** is a powerful costing strategy that uses estimates of particle velocities in the neighbourhood of p1 to predict its likely location in the next frame. A two-dimensional polynomial, fitted in a least squares sense, is computed for each velocity component, based on the velocity estimates of a number of particles that are p1's nearest neighbours in frame 1. The user selects the order of the polynomial to be used - up to a maximum of 4 - and the number of neighbours used to compute the least squares estimates. If the user selects an insufficient number of nearest neighbours, or the number of velocity estimates available (due to neighbouring particles being unmatched) are insufficient, for the order selected, *FluidStream* automatically reduces the order until the polynomials can be computed. The polynomial for each velocity component is evaluated at the position of p1 and this estimated velocity vector is used to predict the location of p1 in frame 2.

The cost associated with a particular match is calculated using equation (4.7)

$$C_{12} = \frac{\left|\mathbf{r}_{pred} - \mathbf{r}_{p_2}\right|}{\left|\mathbf{r}_{pred} - \mathbf{r}_{p_1}\right|} \qquad (4.7)$$

- $\mathbf{r}_{pred}$ – is the predicted location of p1 in frame 2 using the velocity estimate described above
- $\mathbf{r}_{p2}$ – the location of p2 in frame 2.
- $\mathbf{r}_{p1}$ – the location of p1 in frame 1.

The user has a number of parameters that can modify the performance of this costing. To ensure that only particles that are

close to the particle of interest are used in the least squares velocity estimate a maximum distance can be specified. Any of the nearest neighbours lying further than this distance from p1 are excluded from the least squares calculations. Particles lying far from p1 are also likely to have velocities that are less representative of the velocity of p1 than those close to it. The contribution of the velocity estimate of each surrounding particle is weighted in the least squares computations, based on its distance from p1. The weight factor decreases linearly from 1 for a particle at the same location as p1, to a value, specified by the user, for the furthest particle from p1.

The magnitude of the predicted displacement of p1 from frame 1 to frame 2 is used to normalise the cost. If this displacement falls below a default length, specified by the user, then the default length is used in the normalisation instead.

4.4.3.2 Local velocity costing

The **local velocity costing** is similar to the least squares velocity costing in that estimates of particle velocities in the neighbourhood of p1 are used to predict its likely location in the next frame. All particles, in a user specified rectangle centred on p1, that have a velocity estimate available due to a particle match in the previous or next frame, are used to calculate a representative velocity for p1. This representative velocity can be either the average of these particle velocities or their median. When an average of the velocities of the surrounding particles is used the contribution of each velocity to the average is determined by a weight factor. The weight factor decreases linearly from 1 for a particle at the same location as p1, to 0 for a particle located in one of the corners of the rectangle.

Equation (4.7) is again used to calculate the cost of a match, and a default normalization length, as described in section 4.4.3.1, is defined.

4.4.3.3 Magnitude velocity costing

The **magnitude velocity costing** attempts to minimise the change in velocity magnitude between frames. The difference between the velocity magnitude of p1, based on its match to p2, is compared to a second velocity magnitude estimate. This velocity magnitude may be calculated using a match to p1 in the previous frame, a match to p2

in the next frame, or both. The difference between these two velocity magnitude estimates is normalised by their average to yield the cost.

Thus

$$C_{12} = \frac{\left|\left|\mathbf{r}_{p_2} - \mathbf{r}_{p_1}\right| - \left|\Delta\mathbf{r}\right|\right|}{\frac{1}{2}\left(\left|\mathbf{r}_{p_2} - \mathbf{r}_{p_1}\right| + \left|\Delta\mathbf{r}\right|\right)} \tag{4.8}$$

- $\Delta\mathbf{r}$ – the estimate of the displacement based on previous or past matches.
- $\mathbf{r}_{p2}$ – the location of p2 in frame 2.
- $\mathbf{r}_{p1}$ – the location of p1 in frame 1.

4.4.3.4 Path length costing

The **path length costing** is designed to penalise short paths as these are generally seen as unreliable. The cost is given by

$$C_{12} = \frac{1}{particles\,in\,path} \tag{4.9}$$

- *particles* in path – is the number of particles in the path created by matching p1 to p2.

4.4.3.5 Polynomial velocity costing

The **polynomial velocity costing** is similar to the other velocity based costings. A particle path is constructed using p1, p2 and all of the particles linked to p1 in previous frames, and p2 in subsequent frames. Polynomials are fitted, in a least squares sense, to the *x* and *y* coordinates of particles in the path, as functions of the frame number. The cost is calculated from the difference between the actual particle locations and their locations as predicted by the fitted polynomial. It is assumed that the correct particle path should be relatively smooth and therefore even a relatively low order polynomial should be able to accurately represent the path.

The cost is given by

$$C_{12} = \frac{fit\ error}{average\ displacement} \tag{4.10}$$

- *fit error* – the error between the positions of the particles in the path, and the positions of the particles predicted by the polynomial path fit. It can be defined in two ways. Either it is the total fit error based on all particles in the path (this is the root mean square error of the *x* and *y* coordinate fits for each particle), or it is a local error based solely on the fit error for particles p1 and p2. The user chooses which of these definitions to use.
- *average displacement* – the average displacement of the particle from frame to frame is used to normalise the cost.

The order of the polynomial used to fit the particle path is given by

$$order = \frac{path length}{2} \tag{4.11}$$

4.4.3.6 Recent acceleration costing

The **recent acceleration costing** is similar to the recent velocity costing described in the next section. It assumes that accelerations in the flow are roughly constant. The acceleration of the particle, calculated on the basis of a match between p1 and p2 is compared with an acceleration based on particles linked to p1 in previous frames, particles linked to p2 in subsequent frames, or an average of both if these are available.

Thus the cost is given by

$$C_{12} = \frac{\left| \mathbf{a}_{pred} - \mathbf{a}_{est} \right|}{\frac{1}{2}\left| \mathbf{a}_{pred} + \mathbf{a}_{est} \right|} \tag{4.12}$$

- $\mathbf{a}_{pred}$ – the predicted acceleration based on the match between p1 and p2.
- $\mathbf{a}_{est}$ – the estimated acceleration as described above.

4.4.3.7 Recent velocity costing

The **recent velocity costing** strategy was one of the earliest ideas in PTV analysis. It assumes that the accelerations in the flow are small and therefore if the velocity of the particle is known for the current frame, then the location of the particle in the next frame can be estimated from this velocity.

The estimate for the velocity can be based on a match to p1 in the previous frame, a match to p2 in the next frame, or both if they are available.

The cost is calculated using equation (4.7) where the predicted position is based on the velocity estimate just described.

4.4.3.8 Space average acceleration costing

The space average acceleration costing is similar to the local velocity costing. Equation (4.12) is again used to calculate the cost but now the estimated acceleration is based on the average acceleration of the particles lying within a rectangle centred on p1. The accelerations are weighted in the average calculation by the inverse of their distance from p1.

4.4.3.9 Time average velocity costing

The **time average velocity costing** is similar to the recent velocity costing. The difference between the two is the way in which the velocity of p1 is estimated. For the time average velocity costing the velocity is a weighted average of the velocity throughout p1's time history. A particle path is constructed using all of the particles linked to p1 in previous frames, and p2 in subsequent frames, and the velocities of the particles comprising this path are used in the weighted average.

4.5 Guidelines

The selection of the best costing strategy to use for a particular flow tends to be guided by experience. However, the applicability of most of the costings is relatively general so this section will be provide some general advice on the strengths and weaknesses of the various costings. [Note: The author has attempted to develop a clear dimensionless framework which can be used to classify the performance of the various costing strategies. This has been done for a number, but not all, of the strategies available in *FluidStream*. Further information on this framework can be obtained from the author.]

4.5.1 State based costings

For sparsely seeded flows the distance costing performs well. It is important to note that a sparsely seeded flow is defined to be one where the typical inter-particle spacing is large compared to the typical particle displacement between frames. This definition is most easily stated in non-dimensional form as follows

$$\rho_d = \frac{U\Delta t}{\left(1/\rho\right)^{1/2}} \qquad (4.13)$$

- U – an estimate of the velocity magnitude for the flow
- Δt – the time step between frames
- ρ – the density of particles in the frame.

This type of flow is difficult to handle with the PIV method, but can arise easily in practice. An example is the tracking of a small number (order of *100*) of surface drogues.

As the particle seeding density increases the performance of the distance costing begins to degrade. For these flows the adjacency and correlation costings are excellent performers. These two costings are closely related and their performance is likely to be similar. However, the computational speed of the adjacency costing can be up to an order of magnitude greater than the correlation costing and so it should probably be the costing of choice. The performance of both of these costings is affected by the presence of strong shear in the flow. However this is also the case with a typical PIV system.

The remaining state based costings are of limited use. In an ideal flow the intensity, or particle size, may remain relatively constant, but neither characteristic of a particle is unique and therefore it is a poor basis upon which to base a costing. The centre of mass is an inferior cousin of the adjacency costing.

4.5.2 Matching based costings

Of the matching based costings the local velocity costing is the most flexible and useful. It is considerably less sensitive to incorrect matches than the recent velocity, polynomial velocity and time averaged velocity costings due to the fact that it is using velocities from a number of particles in its estimate of the velocity of p1. The least squares velocity costing is comparable in performance to the local velocity costing and may, in fact, perform better when velocity gradients are strong and the current matches are of high quality.

State based costings, even when performing well, will frequently generate some incorrect matches. The local velocity costing is effective at removing these erroneous matches and replacing them with correct ones if they exist.

The other velocity based costings can perform well when the current matches are almost error free. In this case these costings can help to extend particle paths and improve the overall percentage of correct matches.

Theoretical analyses suggest that the equivalent acceleration based costings have the potential to perform as well if not better than the velocity based costings. However, the practicalities of real experimental data ensure that the particle accelerations are considerably less accurate than their velocities due to the fact that they are obtained from higher derivatives of the displacement with respect to time. Therefore for most practical flows the acceleration based costings are of limited use.

The path length velocity is of limited use by itself. However, it can be a simple way of ensuring that only long particle paths are used in the velocity field estimates.

4.5.3 Summary

The most useful costings are the distance, adjacency, correlation, local velocity and least square velocity costings, with the recent and polynomial velocity costings also having some degree of applicability. However each flow needs to be considered separately and some creative ways of using some of the other costings may be discovered.

We have not attempted here to tackle the more difficult problem of considering the performance of multiple costings simultaneously. Our practice has generally involved the sequential application of a number of costings - each subsequent costing building on the performance of those before.

4.6 Residual optimisation

The focus of discussion up to this point has been the matching of particles between frames using a variety of costing strategies. *FluidStream* provides the user with two alternative processes to the

full optimization process already described. The first of these, **residual optimisation**, is discussed in this section while the second, **cleanup**, is explained in the next.

Residual optimisation is a limited optimisation process that works only with particles that are currently unmatched. Effectively all matched particles are removed from the optimisation process and the defined analysis applies only to those that remain. This process can be useful in that it allows the user to avoid compromising a set of high quality matches while still attempting to improve the overall result of the matching process.

4.7 Cleanup process

The purpose of a cleanup is to remove incorrect matches from the clip by selecting one or more costing strategies that can be used to evaluate the validity of a match. If a match incurs a cost in excess of the MMC, then the cleanup removes that match. New matches are not created. Thus a cleanup performs a post-processing role.

Optimisation parameters, such as the search window, are not relevant to the cleanup process, as new candidate matches are not considered during the process.

4.8 Video clip graph view revisited

At times the user may wish to obtain more detailed information about the matching process. The video clip graph view provides a number of features that gives the user access to internal details of the process.

The user can select particles in the flow by drawing regions on the graph view frame display. By double clicking inside a region the user can view details of the particles within the region. This view is known as the **match details view**. A menu within this view allows the user to display information about the candidate matches for any particle within the region and the costs, calculated from any of the defined PTV analyses, for each possible match. It is also possible to display details of the complete path of matched particles for any particle within the region.

Full details of the matching details view and its associated views can be found in the **User's Guide**.

4.9 Summary

This chapter has introduced the particle matching process in some detail. The theoretical foundations of the optimisation process have been covered along with some practical implementation issues. The various costing strategies, that lie at the heart of the PTV system, have been described.

5 Velocity Field Calculation

5.1 Interpolation requirement

One of the limitations of particle tracking velocimetry is that velocity estimates are located at the positions of the particles in the flow, and these particles tend to be randomly distributed. To be able to look at the time history of the velocity at a point in the flow, for example, the particle velocity field must be interpolated at the point of interest. Therefore the typical final stage of a PTV analysis is to interpolate the particle velocities onto a regular rectangular grid.

The remainder of this chapter looks at this process in more detail.

5.2 Interpolation process

The interpolation process is performed by a **field interpolator** object. The velocity field time series produced by the interpolation process is simply a sequence of two-dimensional velocity fields, defined on a regular grid, where each velocity field corresponds to a frame in the original video clip. Once created the velocity field time series object is displayed in the *FluidStream* time series list view.

Previously saved velocity field time series objects can be opened under the *FluidStream Velocity* menu. Files in which these objects are saved carry a *.vel7* extension. If the velocity field is saved in a text file its file extension is *.vtx7*.

The interpolation process has the following steps:

Velocity calculation

The velocity of each particle, in each frame, must be calculated. Particles that are matched to other particles in the previous or subsequent frame are able to have velocities calculated for them, while particles that are unmatched cannot participate in the process. The velocity calculation can be performed using a **forward difference** approximation (if the particle is matched in the next

frame), a **backward difference** approximation (if the particle is matched in the previous frame) or a **central difference** approximation (if the particle is matched in both frames). By basing the interpolation on only those particles that possess two matches the overall accuracy of the velocity estimates may improve. However, this advantage may be counterbalanced somewhat by the reduced number of particles that can contribute to the interpolation.

If a match exists in the previous frame only, then the velocity is calculated using a simple backward difference scheme

$$\mathbf{u}_{ij} = \frac{\mathbf{r}_{ij} - \mathbf{r}_{kj-1}}{\Delta t} \qquad (5.1)$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{ij}$ – the position vector of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{kj-1}$ – the position vector of the k^{th} particle in the $j\text{-}1^{th}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

Δt – the time step between frames.

If a match exists in the next frame only, then the velocity is calculated using a simple forward difference scheme

$$\mathbf{u}_{ij} = \frac{\mathbf{r}_{kj+1} - \mathbf{r}_{ij}}{\Delta t} \qquad (5.2)$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{ij}$ – the position vector of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{kj+1}$ – the position vector of the k^{th} particle in the $j+1^{th}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

Δt – the time step between frames.

If matches exist in both frames then a central difference scheme is used.

$$\mathbf{u}_{ij} = \frac{\mathbf{r}_{mj+1} - \mathbf{r}_{kj-1}}{2\Delta t} \tag{5.3}$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{mj+1}$ – the position vector of the m^{th} particle in the $j+1^{th}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

$\mathbf{r}_{kj-1}$ – the position vector of the k^{th} particle in the $j-1^{th}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

Δt – the time step between frames.

Another possibility is to use more particles than just those in the previous and subsequent frames. A particle path can be constructed from the particles, linked to the particle of interest, through matches in past and future frames. A polynomial least squares fit to the positions of these particles can be differentiated to obtain the velocity. The order of the polynomial is given in equation (4.11).

In this case the velocity is given by

$$\mathbf{u}_{ij} = \left.\frac{d\mathbf{r}}{dt}\right|_{t_j} \tag{5.4}$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}(t)$ – is the polynomial fit to the particle path

t_j – is the time of the j^{th} frame.

No matter which method is used to calculate the velocity of a particle, erroneous matches will adversely affect the calculated velocity field. The elimination of incorrect matches must remain the principal goal of the analyst. Sometimes this will come at the expense of the total number of matches made. In general, missed matches, compromise the velocity field to a far lesser degree than incorrect matches.

Grid specification

For the interpolation process to proceed, a regular, rectangular grid must be specified. Velocities will be interpolated onto the nodes of this grid. The selection of the grid must be an informed decision. Very fine grids, to resolve small-scale structures in the flow, may seem highly desirable. However, if the density of the particles themselves does not provide sufficient resolution to identify these structures a fine grid will not help.

Another consideration in grid selection is the issue of accuracy near the frame edges. To be more specific, accuracy near the edge of the particle field in each frame. A grid point that lies outside the region covered by the particles (actually, outside the triangulation of the particles – see below) will require extrapolation to determine its velocity. In addition, particle matching near frame boundaries is also error prone, due to the exit and entry of particles over time. Therefore, it is recommended that velocity information is not sought near the boundaries of the region covered by particles.

Interpolation

A number of schemes exist for interpolating a function, defined at randomly located points in the plane, and three different options are provided. These are denoted

- Triangle based
- Least squares
- Binned

The triangle based approach adopts the method of Cline and Renka (1984) that produces a triangulation of the particles in a frame and uses this triangulation as the basis for the interpolation. This triangulation, known as a **Delaunay** or **Thiessen triangulation**, has a number of special properties, the most important of which is that the minimum interior angle in two adjacent triangles is maximised. This has a number of corollaries, but from a practical point of view it simply means that the triangles tend not to be long and thin. Thus interpolations within the triangle will be based on three particles that are “close together”.

To estimate the velocity at a grid point its containing triangle is found, and the velocity is interpolated linearly from the velocities of

the particles at the triangle's corners. This interpolation is based on that used for a triangular element in the finite element method (FEM). If a grid point is located outside the triangulation the velocity at that point is left undefined. In addition the user is at liberty to specify the maximum size of triangle that may be used to interpolate the velocity data. If a grid point is located in a triangle that has dimensions larger than those set by the user then the velocity will be left undefined at that grid point.

The least squares approach fits a first, second or third order, two-dimensional, polynomial function to the velocities of particles neighbouring the point of interest, using the local method of Renka and Cline (1984). This function is fitted in a least squares sense, and the user is at liberty to chose the number of particles that participate in this fitting process. If the user chooses to extrapolate their velocity data beyond the triangulation described above the least squares method must be used.

The binned approach follows the method suggested by Cowen and Monismith (1997). A rectangular bin, centred on each grid point, is defined by the user. The velocities of particles that fall within each bin are averaged in order to estimate the velocity at the associated grid point. This is a zeroth order interpolation scheme, but if particle densities are high, the result should be robust. If no particles lie within a particular bin in one frame, the velocity at that grid point is left undefined.

Partial derivatives of the velocities are also required if derived flow quantities such as vorticity are to be evaluated. The user has, again, a range of options for the calculations of the partial derivatives. These are

- Grid finite difference
- Triangle least squares
- Grid least squares
- Triangle constant interpolation

The grid finite difference approach uses a finite difference approximation to the velocity derivatives at each grid point using the velocities interpolated onto the rectangular grid. The user can select the order of the finite difference scheme used. If a second order scheme is selected then derivatives are only available at grid points where velocities are defined on either side.

The triangle least squares approach uses the least squares approach described above to fit a two-dimensional polynomial to the particle velocities surrounding each particle in the frame. This function then provides first order derivative estimates at each particle and the partial derivatives at each grid point are calculated using triangulation based interpolation of the derivatives.

The grid least squares method uses the same least squares approach but fits the function at the grid points themselves, thus avoiding the use of the triangulation altogether.

The triangle constant interpolation method uses the triangulation technique and fits a first order two-dimensional function to the velocities at the three vertices of the triangle containing the grid point. The first derivatives of this function are used as the estimates for the partial derivatives at the grid point.

The user is at liberty to choose any combination of these various interpolation and partial derivative estimation methods.

The accuracy of the interpolation process is critical to the PTV analysis, and the limitations of the algorithms used here should be borne in mind. For smoothly varying flows, with high particle density, none of the interpolation process are likely to introduce significant error compared to other areas of uncertainty, such as particle identification and particle matching. However, for flows with significant gradients the legitimacy of the interpolated fields should always be critically reviewed.

5.3 Field calculators

The result of the interpolation is a velocity field time series that is visible to the user in the *FluidStream* time series list view. As they stand, this sequence of two-dimensional velocity fields is difficult for the user to view and digest. To alleviate this problem each velocity field time series provides a number of **field calculators**. A field calculator represents some field derived from the velocity field time series, and it provides the user with a vehicle for calculating this field on some projection of the three-dimensional space (x, y and t) over which the velocity field is defined.

The concept is most easily understood by an example. One of the field calculators provided is the vorticity. The **manager view** of the

vorticity calculator allows the user to calculate a range of sub-fields. For example, the user may wish to calculate, and plot, the time-averaged vorticity field, or perhaps the vorticity field's time variation at a fixed point in space.

In all, twenty one different calculators are provided, and each of these can calculate thirty different projections of the three-dimensional calculated field. The field calculators are given in table 5.1

Symbol	**Description**
(u,v)	the instantaneous velocity vector
(u',v')	the fluctuating velocity vector. The time average velocity vector, calculated from the entire time series is subtracted from (u,v) to generate this vector.
|uv|	the magnitude of the instantaneous velocity vector
u	the x component of the instantaneous velocity vector
v	the y component of the instantaneous velocity vector
(du/dt,dv/dt)	the local acceleration vector
coverage	the coverage is defined to be 1 at a grid point at which the velocity is defined and 0 if it is not defined.
KE	the 2D kinetic energy based on the instantaneous velocity field.
vorticity	the vorticity field
div(u,v)	the divergence of the instantaneous velocity field
strain rate	the strain rate of the instantaneous velocity field
TKE	the 2D turbulent kinetic energy. The time average of this field gives the 2D TKE per unit mass.
-u'v'	the negative correlation between the fluctuating velocity components. The time

Symbol	Description
	average of this field gives the turbulent shear stress per unit mass
u'u'	the intensity of the fluctuating x velocity component.
v'v'	the intensity of the fluctuating y velocity component.
u'	the x component of the fluctuating velocity field
v'	the y component of the fluctuating velocity field
du/dx	partial derivative of u with respect to x.
du/dy	partial derivative of u with respect to y.
dv/dx	partial derivative of v with respect to x.
dv/dy	partial derivative of v with respect to y.

Table 5.1 The list of available velocity field calculators.

Full details of these calculators are given in the **User's Guide**.

Users should bear in mind the accuracy limitations of many of these derived fields. In particular those fields based on turbulence statistics or velocity gradients, must be interpreted with care.

5.4 Velocity field transformations

Velocity field time series, in their raw form, may not provide the information required by the experimentalist. For example, consider a fluid flow that is steady in a moving frame of reference, captured by a stationary camera. The experimentalist would ideally wish to view the data in the moving frame. *FluidStream* provides a number of velocity field transformations that allow the user to transform their velocity field data in some way. Table 5.2 provides a brief summary of the available transformations. See the **User's Guide** for more detailed information.

Name	Description
Linear	Allows the user to scale and offset the velocity field and the x, y and t coordinates. Useful for converting data to dimensionless form.
Subset	Allows the user to extract a subset of the velocity field.
Swap x	Reflects the data along a vertical centre line.
Swap y	Reflects the data along a horizontal centre line.
Translate fields using constant velocity	Shifts the 2D velocity fields by a constant velocity so that the spatial domain becomes stretched in the direction of motion.
Translate fields using file displacements	Shifts the 2D velocity fields by displacements stored in a text file. Similar to the transformation above but this allows for non-uniform displacements.
Average fields in time	Performs a partial time average. The user specifies the number of frames over which the time average is computed. The result is a time series with fewer frames and a larger time step – each frame now representing an average.
Increment velocities using file velocities	The velocity field at each time step is incremented by a velocity stored in a text file. The text file may contain as many velocity vectors as there are frames in the time series, and therefore the velocity field in each frame can be incremented by a different vector.

Table 5.2 The list of velocity field times series transformations available in *FluidStream*.

5.5 Combined velocity fields

FluidStream also provides mechanisms for combining velocity field time series. This may be useful, for example, when constraints on computer memory or other experimental factors, limit the number of video frames that can be captured in one experimental run. Concatenating a number of velocity time series allows the user to artificially construct a longer time series from a number of shorter runs.

The three combination mechanisms available are listed in table 5.3.

Name	Description
Concatenation	A number of velocity field time series are concatenated (ie strung together). The user can choose spatial offsets for each time series relative to the first.
Overlay	A number of velocity field time series are overlaid. The user can select spatial and temporal offsets for each series relative to the first and also an overlay rule. – how the velocity fields are combined if more than one time series provides the velocity at a particular time and place.
Subtraction	Two velocity fields are subtracted from one another. This allows the user to compare velocity fields of the same flow generated perhaps using different interpolation schemes or different particle identification parameters. The resulting velocities can be absolute, or relative to the original velocities (ie fractional errors).

Table 5.3 The list of velocity field times series combinations available in *FluidStream*.

5.6 Density fields

While the density of particles in the flow is not generally a quantity of interest to someone performing particle tracking velocimetry (in fact, a uniform particle distribution is generally desirable) this field can be calculated by *FluidStream*.

The user defines a grid on which the density is to be evaluated. The number of particles lying in the rectangle surrounding each grid point is evaluated and divided by the area of the rectangle. Derivatives of the density at each grid point are calculated using the finite difference method described for the interpolation of the velocity field.

Once evaluated the density field time series is displayed in the *FluidStream* time series list view.

Each density field time series provides eight field calculators. These are give in table 5.4.

Symbol	**Description**
rho	the instantaneous density field
rho’	rho’ – the fluctuating density field. The time average density, calculated from the entire time series, is subtracted from rho to generate this field.
rho’rho’	the intensity of the fluctuating density.
grad(rho)	the gradient of the density field
grad(rho)|	the magnitude of the gradient of the density field
drho/dx	partial derivative of density with respect to x
drho/dy	partial derivative of density with respect to y
drho/dt	partial derivative of the density with respect to t

Table 5.4 The list of available density field calculators.

5.7 Summary

The final step in a particle tracking velocimetry system is the calculation of the velocity field on a regular grid. This chapter has summarised *FluidStream*’s implementation of this process, highlighting areas that will affect the accuracy of the derived velocity fields. The field calculators, provided with each velocity field, offer a powerful analysis environment in which the user can explore their experimental results. The calculation of the particle density field has also been discussed.

6 Practical Guide

The previous chapters have described the various *FluidStream* subsystems. In this final chapter a practical guide to the operation of *FluidStream* is provided. This guide will lay out the steps a user would typically follow when performing a PTV analysis using *FluidStream*.

6.1 Image capture

The first step in this process is the acquisition of high quality flow images. A number of the issues involved in this step have been introduced in Chapter 1 where camera selection, light sources and particle material have been discussed. This step is outside the *FluidStream* system, but a number of observations can be made about the data ideally produced during image capture.

- The particles in the video images should be clearly differentiable from the background and any noise present in the image.
- The particles must correspond to regions of high light intensity on a dark background.
- The particle density is at the discretion of the user. However, in general, for high quality velocity results, a density of 1000-3000 particles per frame is recommended.
- Particle density should be as uniform as possible throughout the images.
- Images must be converted into a sequence of JPEG, GIF, TIF, BMP or PNG files.

6.2 Video clip creation

FluidStream enters the analysis process at this point. The experimental images are loaded into *FluidStream* and a video clip is produced. The key step here is the identification of particles and it is important that this part of the analysis is considered carefully.

The user must choose which of the three available algorithms they will use for this identification process (see section 3.3). The user should experiment with the various algorithmic parameters in order to obtain the best set of particles. The best approach to checking the quality of the data is to look at the match summary and graph views

of the video clip once it has been created. The match summary view allows the user to check that the number of particles per frame is consistent with their expectations, while the graph view allows the user to display many frames at once in order to discern the particle paths within the flow. If either of these tests appears to be unsuccessful then the original images should be reanalysed with different algorithmic parameters.

Perhaps one of the most common pitfalls here is attempting to increase the number of particles in each frame by lowering the intensity threshold parameters. Frequently this leads to the identification of particles that are very faint, or non-existent, and the clip's graph view will typically indicate a large numbers of particles leaving and entering the flow at each time step. A better solution is to work with lower particle densities or to improve the lighting/image capture system.

6.3 Particle matching

Once the video clip has been produced the user can proceed to the PTV analysis itself (see chapter 4). However, it is recommended that it is worthwhile becoming familiar with the data through the features available in the video clip graph view before a full PTV analysis is performed. This familiarity can lead the user to identification of potential problem areas within the flow, but more importantly may guide the user to the most sensible choice of costing strategies to use in the PTV analysis.

As with the particle identification process, it is important to experiment with the particle matching algorithm. This is best done by taking a small subset of the total number of frames, that are representative of the clip as a whole, and exploring the effectiveness of different costing strategies in analysing these frames.

Typically a number of PTV analyses will be defined for a clip and executed sequentially. It is possible for some flows that PTV analyses that work with only a subset of the flow domain will be defined. The user has considerable flexibility in this regard and it is recommended that he/she take advantage of this in order to obtain the best results.

On the completion of the PTV analysis it is crucial that the final particle matches are critically analysed. This is most easily done in

the match summary and graph views. The graph view, when a number of frames are displayed simultaneously, normally makes it possible to discern erroneous particle matches. The match summary view indicates the percentage of the particles that are matched. Ideally, for good quality velocity data this figure should be around 80% depending on the particle density.

6.4 Velocity field creation

Once the particle matches have been finalised the last remaining step in the process is the generation of the velocity field (see chapter 5). This step is more one of fine-tuning the data than substantially changing it. The different interpolation schemes have different strengths and weaknesses but, unless unusual parameters are selected for this process, the results from the different schemes should not vary greatly. Certainly the velocity fields should be checked to ensure that they match the flow structure that can be identified from the particle matches in the video clip graph view.

As discussed in the text, care should always be taken when working with turbulence statistics and velocity gradients, as these quantities carry with them the largest error.

6.5 Summary

This chapter has provided a very brief overview of a PTV analysis performed using *FluidStream*. Details about each step in the process can be obtained from the relevant chapters and the **User's Guide**.

7 Conclusions

This manual has presented an overview of the theory and design of *FluidStream*, a component of the *Streams* library developed in the Department of Civil Engineering at the University of Canterbury.

The following areas have been covered

- an overview of a PTV system,
- practical issues of lighting and image capture,
- operational and system issues, computer system requirements, and the user interface model,
- image processing and particle identification, leading to video clips created from image sequences,
- video clip creation from theoretical models,
- particle matching and its optimisation, and
- velocity field interpolation, and the generation of derived fields.

Nearly all of the specifics relating to the operation of *FluidStream* are included in the **User's Guide**. It includes detailed information on parameter definitions and selection, as well as descriptions of *FluidStream* objects and their associated views.

The user will find that effective use of *FluidStream* requires experience in its operation, as well as a firm understanding of its various components. We would encourage the user to spend some time exploring *FluidStream*'s features within the environment provided by the theoretically derived fluid flows.

8 References

Bertsekas D P 1992, “Auction algorithms for network flow problems: a tutorial introduction”, *Report LIDS – P – 2108*, Laboratory for Information and Decision systems, MIT, Cambridge, Mass pp 54.

Blackett S A 1994, “Particle tracking velocimetry”, *ME Thesis*, University of Auckland, Auckland, New Zealand, pp 86.

Cline A K and Renka R L, 1984, “A storage-efficient method for construction of a Thiessen triangulation”, *Rocky Mountain Journal of Mathematics*, **14**, No. 1, pp 119-139.

Cowen E A and Monismith S G, 1997, “A hybrid digital particle tracking velocimetry technique”, *Experiments in Fluids*, 22, pp 199-211.

Nokes R I, 2002, “Interface proxy classes in object-oriented user interface design – an extension to the model-view architecture”, *New Zealand Journal of Applied Computing and Information Technology*, 6, 1 pp 37-44.

Raffel M, Willert C, Kompenhans J, 1998, “Particle Image Velocimetry”, *Springer-Verlag*, Berlin, Germany, pp 253.

Renka R L and Cline A K, 1984, “A triangle-based C1 interpolation method”, *Rocky Mountain Journal of Mathematics*, **14**, No. 1, pp 223-237.

Smits A J and Lim T T, 2000, “Flow visualisation”, *Imperial College Press*, London, UK, pp 396.

Appendix A
Installation and Execution

The *FluidStream* application is deployed as a ZIP file, either on CD or downloadable from the web (contact the designer for information on this). Different ZIP files are provided for Macintosh and Windows systems and the installation process is slightly different on these systems.

A.1 Installation under Windows

The ZIP file required for installation on a Windows computer is named

FS700_Win.zip

To install *FluidStream* on a *Windows* computer perform the following steps:

1. Create a directory in the root directory of the computer C: drive called *StreamsLibrary*. Within this directory create a second directory called *FluidStream7.00* and extract the contents of *FS700_Win.zip* into this directory. The *FluidStream7.00* directory should contain the following 4 files:

- FluidStream.jar – the java archive file that contains the application
- FluidStream7.00 – the shortcut for running the application
- java.exe – the java interpreter called by the shortcut to run *FluidStream*.
- FluidStream32x32.ico – the icon used for the *FluidStream* shortcut.

and a directory containing the PDF versions of the software manuals.

The shortcut for running *FluidStream* can be copied to any location you wish.

2. Edit the target within the shortcut to allow for your system configuration (if required). The target is accessed through the shortcut's popup menu. Select Properties and go to the Shortcut tab.

You may need to change the file attributes on the General tab so that Read-only is switched off. As delivered the target is

```
C:\StreamsLibrary\FluidStream7.00\java.exe –Xms800m –Xmx800m
–Xss32m –classpath
C:\StreamsLibrary\FluidStream7.00\FluidStream.jar FluidStream.Application.FluidStream
```

Provided you leave the .jar and .exe files in the *C:\StreamsLibrary\FluidStream7.00* directory none of the paths need to be changed. If you wish to move them to another directory then the new path should precede the filenames in the target. For example, if you wish to put these files in a directory

C:\PTV

Then the target would become

```
C:\PTV\java.exe –Xms800m –Xmx800m
 –Xss32m –classpath
C:\PTV\FluidStream.jar FluidStream.Application.FluidStream
```

The three parameters following the java.exe set the memory size for the application. The first is the initial heap space, the second the maximum heap space and the third the desired stack space. The figures 800m, 800m and 32m refer to *800Mb*, *800Mb* and *32Mb* respectively. Depending on the RAM installed on your system you may need to change these settings. Experiment with these so that the application runs with the maximum settings. A minimum of *32Mb* for the stack space, with the rest of the memory allocated to the heap, should be a good initial setting.

It is also possible that you already have a later version of java installed on your computer. If this is the case then there is no reason to remove this version and install version 1.4 if you replace the file java.exe in the *FluidStream7.00* directory with the same file provided with the installed version of java. This file can be located in a subdirectory of the Java directory in C:\Program Files. A search of the C:\ drive is probably the easiest way to locate this file.

3. *FluidStream* is executed by double-clicking the *FluidStream7.00* shortcut icon.

A.2 Installation on a Macintosh

The ZIP file required for installation on a Macintosh computer is named

FS700_Mac.zip

To install *FluidStream* on a *Macintosh* computer perform the following steps:

1. Create a folder in the root directory of the hard drive called *StreamsLibrary*. Within this folder create a second folder called *FluidStream7.00* and extract the contents of *FS700_Mac.zip* into this folder. The *FluidStream7.00* folder should contain the following 2 files:

- FluidStream.jar – the java archive file that contains the application
- FluidStream7.00.app – the Apple script for running the application

and a directory containing the PDF versions of the software manuals.

The Apple script file for running *FluidStream* can be copied to any location you wish.

2. Edit the command within the Apple script file (*FluidStream7.00.app*) to allow for your system configuration (if required). Use the Apple Script editor to do this. As delivered the script file should contain a single command

```
do shell script java -jar –Xms800m –Xmx800m
–Xss32m //StreamsLibrary/FluidStream7.00/FluidStream.jar
FluidStream.Application.FluidStream
```

Provided you leave the .jar file in the *//StreamsLibrary/FluidStream7.00* folder the path need not be changed. If you wish to move it to another folder then the new path should precede the jar filename in the script command. For example, if you wish to move the .jar file to a folder called

//PTV

then the command would become

```
do shell script java -jar –Xms800m –Xmx800m
–Xss32m //PTV/FluidStream.jar FluidStream.Application.FluidStream
```

The three parameters following the -jar setting specify the memory size for the application. The first is the initial heap space, the second the maximum heap space and the third the desired stack space. The figures 800m, 800m and 32m refer to *800Mb*, *800Mb* and *32Mb* respectively. Depending on the RAM installed on your system you may need to change these settings. Experiment with these so that the application runs with the maximum settings. A minimum of *32Mb* for the stack space, with the rest of the memory allocated to the heap, should be a good initial setting.

3. *FluidStream* is executed by double-clicking the *FluidStream7.00.app* icon.

Appendix B Registration

FluidStream must be registered before it can be run. A registration key can be requested from Dr Roger Nokes.

When *FluidStream* is run for the first time a registration dialog box is displayed that prompts the user to electronically agree to the conditions of software use and to enter a 20 character registration key. This key is checked by the software, and if valid, the application is launched. Subsequent launches of the software do not require the registration key to be reentered.

Part of the registration key is the registration expiry date. If the application is run after this date the user will be prompted to reregister the software. The expiry date can be viewed in the application parameters dialog box under the *Application* menu of the main window. The date on which the software was registered is also displayed in this dialog box.